SPECIFICATION

Electronic Version 1.2.8

Stylesheet Version 1.0

A METHOD FOR THE AUTOMATIC GENERATION OF COMPUTER PROGRAMS WHICH INTERACT WITH EXISTING OBJECTS

Background of Invention

[0001] The present invention relates generally to a system and method for creating and executing application programs and other software and, particularly, to a system for automating the task of developing application programs which interact with existing objects.

[0002] Today, complex, mission critical software has become the rule both in the business and scientific communities. Typically, such programs require communication between one or more objects, such as existing software, in the same or other computers connected in a network. As an end result, sophisticated end-user interfaces, complex reporting capabilities, computation intensive software components and software integration across platforms have become the rule. Such sophistication, in turn, requires sophistication in the maintenance and modification of such software. Software developers, by necessity, must be able to write programming code for multiple platforms; communicate between diverse software programs and objects; and develop interfaces and systems for their integration in a single user interface. When one takes into account that computers in this context mean a broad array of intelligent devices including Internet appliances, cellular phones, hand held organizers and for that matter, household

appliances and automobiles, the difficulty of universal programming and integration becomes readily apparent.

[0003] As is all too familiar to anyone who uses such sophisticated systems, this collection of skills and abilities is both not readily available and, even when available, is no guarantee that their utilization will speedily result in the desired application. Typically, drafting thousands of lines of computer programming code is followed by extensive testing and correction. Historically structured programming has been the norm. That is, various functions are first defined and the program is designed to call such defined functions at appropriate times to effect the overall objective of the application program.

[0004] One approach in solving this problem has been to use "application generators." These are programs which generate source programming code for application programs. Such tools are normally referred to as Rapid Application Development (RAD) tools such as Borland's AppExpert, and Microsoft's AppWizard. In such systems, the software developer provides the tool with initial input for indicating the type of application program desired. The software then uses these inputs to produce complex source programming code for a working program which includes the attributes specified by the user. It does this by using highly structured templates into which the software developer has indicated certain parameters. These slightly modified templates then have to be extensively altered by the software developer to accomplish the desired result. In other words, such programs build a skeleton from standard parts to which the software developer then adds the flesh to produce the desired end result.

[0005]

In contrast to such software generating programs or the even earlier approach of simply writing the program from A to Z, the new paradigm is that of object oriented programming. An object is defined by its external interface and the software developer considers, and for that matter interacts, only with the external interface of the object. However, the software developer must still fully understand the interface in order to take full advantage of the object in its interaction with the program under development. Thus, the software developer must not only be

familiar with the programming language and the program generating application, but with every object with which the desired end product will interact.

[0006] Where the desired program is to be used on several different platforms, the software developer's task may be multiplied by the use of multiple programming languages, programs and objects. Thus, while existing program building software relieves some of the task, the major programming tasks are still left to a software developer able to develop code in one or more programming languages and understand the action of assorted objects with which the program is to interact.

[0007] The resulting application normally resides on each client computer. It may interact with one or more objects which reside on that client computer or on another computer in a network with that client. Thus, the resulting application must contain the complete coding necessary to carry out its function and to interact with the other objects. If the program is a large one, the program not only requires substantial space on each computer where it resides, but each modification of the application may require substantial time for the original or new versions to be distributed through a network to each of the client computers.

[0008] What has been lacking, up to the present, is a computer program development tool which generates a working program from a mere indication of desired function, including automatically analyzing and integrating with any desired objects without the software developer having extensive knowledge as to the nature of the object interface, nor less the object programming. In other words a program that explores the objects with which it is to interact, not only eliminating the need to know the computer language and interface of the object but even the human language it displays in its screen. Equally lacking is a cross platform development tool which minimizes the size of the resulting application. In other words, what is inherently desirable, but missing from the existing systems, is a tool for automatically developing a working, cross-platform and cross-human language program of minimum size which automatically interacts with desired existing objects.

Summary of Invention

[0009] The present invention goes beyond the present object-oriented approach by, in essence, automatically studying particular existing objects and automatically developing an appropriate interface to directly interact with the object or a subset of the object. The querying of the object or objects and structuring of the desired program is done by an editor or development tool (the "Workbench") which would normally reside on the software developer's computer or be accessible by the software developer's computer through a network. The objects which it explores may reside on the software developer's computer, or elsewhere on an intranet or the Internet.

[0010] The Workbench produces a collection of data which delineates the attributes and parameters of the desired application (the "Project"). Through the Workbench's simple graphic interface, a software developer identifies the Project and indicates the external objects (i.e., objects that are defined and created outside of the Workbench) which are to interact with the program under development. The Workbench by querying the external object, i.e., by reflection, discovers the interface to each external object which will be incorporated into the Project and uses this information in the development of the Project. In other words, it learns the methods and fields of the external objects, and by automatically developing a description of them, develops a description of a functioning application (the "Default Project").

[0011] After the Workbench interrogates the external object or objects, the Workbench may be used to refine the resulting Default Project. The software developer may add graphics, change colors, change fonts or other design aspects from that of the Default Project. The software developer may also use the Workbench to add sections of programming code for interaction with external objects. For example, the software developer could add an automatic calculation or a filter to alter an object or generate a new external object to the Project. Equally, the software developer may change the look and feel of the Default Project by such additional programming code. The additional programming code is in an appropriate cross-platform programming language such as Java. Use of the Java programming language to create extensions to elements of the Default Project broaden the

capabilities of the resulting application in comparison to the cookie cutter approach of current RAD tools.

[0012] After development of the Default Project, and its editing and tweaking using the Workbench is completed, the software developer finalizes the Project, i.e., creates a "Built Project." In finalization, the Workbench creates a compressed storage space to hold the Project, compiles all of the programming code that the software developer has specified during editing, loads the Project specification into memory, serializes the Project specification while removing Project attributes that only have value to the development process, and adds items such as graphic files and help files. If the resulting application is to run in the Sun Microsystems' Java Runtime Environment (JRE), the storage space would be in the in the form of a Java Archive file (JAR).

[0013] The information in the Project is used to build the desired application by an interpretive program (the "Runtime") which is not specific to any one Project. The application produced by the Runtime operates independently of the Runtime, i.e., after generation of the application, the Runtime plays no part in the operation of the resulting application.

[0014] If the application is to run in the JRE, the runtime would also be written in Java programming language and would be stored as a JAR. It thus could take advantage of the JRE already residing on most computers. The Built Project would be stored as a separate JAR. The Built Project would be relatively compact. To distribute an application to an individual computer having access to the Runtime either locally or over a network requires only the JAR for the Built Project, since the JRE would already permanently reside on the user's computer. The advantages for distribution of programs over the Internet are obvious. The resulting programs are significantly smaller, faster, scalable and deployable not only as a standalone application but as web applets.

[0015] As can be seen from the above, to develop a Project, first the Workbench receives the initial instructions from the software developer and queries the external objects which are to interact with the developed program. The Built Project

contains all of the instructions developed in the Workbench and the information specific to the Project as a series of instructions for the Runtime. The Runtime takes the Built Project and using a runtime environment such as the JRE, builds the desired application. The purpose of the Runtime is to reproduce the user interface that was designed using the Workbench and connect that user interface to the specified external objects. Once it performs this function, the Runtime's job is done. There is no direct interaction between the software developer and the Runtime. While the Workbench itself may be a Built Project, the Runtime is, in operation, independent of the Workbench, which however is a necessary part of the development of the Built Project. The end user only directly interacts with the Built Project and the resulting application, not the Workbench nor the Runtime.

[0016] Since the Runtime uses the JRE already residing on most computers today to supply much of the programming code to run the Built Project, even what would be very large programs when constructed by prior art methods can, by skilled use of the Workbench, be reduced in size by 80 or 90 percent.

Brief Description of Drawings

[0017] Fig. 1 is a flow diagram of the overall process of generating a Project;

[0018] Fig. 2 is a flow diagram for automatically generating a Default Project;

[0019] Fig. 3 is the initial screen shot of the Workbench;

[0020] Fig. 4 is a screen shot of the Workbench with the product identity dialog box open;

[0021] Fig. 5 is a screen shot of the Workbench with the connection dialog box open;

[0022] Fig. 6 is a screen shot of the Workbench with the component selection dialog box open;

[0023] Fig. 7 is a flow diagram of the information generation for external objects;

[0024] Fig. 8 is a schematic representation of a default project;

[0025] Fig. 9 is a screen shot of the Workbench with the project dialog box open;F

[0026] Fig. 10 is a screen shot of the Workbench with the project window open;

[0027] Fig. 11 is a screen shot of the Workbench with the new frame dialog box open;

[0028] Fig. 12 is a screen shot of the Workbench with the properties window open;

[0029] Fig. 13 is a screen shot of the Workbench with the new panel box open;

[0030] Fig. 14 is a screen shot of the Workbench with the panel folder expanded;F

[0031] iFg. 15 is a screen shot of the Workbench with the properties information window open;F

[0032] Fig. 16 is a screen shot of the Workbench with the label adding box open;

[0033] Fig. 17 is a screen shot of the Workbench with the editor window open;

[0034] Fig. 18 is a screen shot of the Workbench with the editor window for parameters open;

[0035] Fig. 19 is a screen shot of the Workbench with the visual object dialog box open;

[0036] Fig. 20 is a screen shot of the Workbench with the editor window open and a new button being displayed;

[0037] Fig. 21 is a screen shot of the Workbench with the Java source window open;

[0038] Fig. 22 is a screen shot of the Workbench with the Java class window open;

[0039] Fig. 23 is a printout of the project specification file;

[0040] Fig. 24 is a screen shot of the Workbench with the build project dialog box open;

[0041] Fig. 25 is a schematic representation of a built project;

[0042] Fig. 26 is a schematic representation of a runtime;

[0043] Fig. 27 is a flow diagram of the reflection engine;

[0044] Fig. 28 is a flow diagram of the runtime;

[0045] Fig. 29 is a schematic representation of the relationship between the Server and Client;

[0046] Fig. 30 is a schematic representation of the relationship between the Server and Client;

[0047] Fig. 31 is a schematic representation of the relationship between the Server and Client;

[0048] Fig. 32 is a schematic representation of the local device.

Detailed Description

[0049] The present invention involves two distinct phases. The first phase is the generation of a Project through the use of the Workbench. The Workbench first generates a Default Project based on information supplied by the software developer and, if it is to interact with external objects, identification of such external objects. The resulting Default Project is then edited and finalized by the Workbench to produce a Built Project which can then be distributed to end users for use in conjunction with the Runtime.

[0050] Fig. 1 discloses the overall process of Project generation, while Fig; 2 discloses the details of generating the Default Project which is later edited in the Workbench to complete a Project. In describing the process of Project generation by the Workbench, Fig. 1 and Fig. 2 will be used in conjunction with corresponding screen shots of the Workbench carrying out the steps of the flow diagrams of Fig. 1 and Fig. 2 in developing a Project for calculating compound interest based on an external object.

[0051]

When the Workbench is openedan initial screenappears on the software developer's computer screen. It consists primarily of the Workbench menu bar and tool bar 303. Floating Projects Window 304 and Properties Window 305 are also

displayed. If the software developer is starting a new Project, the software developer will start the Project generation process by so indicating in the file menuor by iconon tool barthat a new Project generation is to be initiated. This, in turn, initiates a wizard which will lead the software developer through the process of developing a Default Project The process for the generation of the Default Project is most clearly seen in Fig. 2.

[0052] As seen in Fig. 4, upon initiation of a new Project wizarda Project Identity Dialog Boxappears which provides text fields for the software developer to name the Projectand to direct where the Project is to be stored In this example, the Project is named "CompoundInterest"and it is to be stored at "C:/My Projects/compoundinterest."If the Project will not be interacting with an external object or the software developer wishes to define the external objects at a later time, the software developer indicates that a Default Base Project should be generated 206 by clicking on the "generate"button The Workbench then generates a formatted file called the Project Specification File. It is a list in a compressed text format of various information about the Project, including the Project name, as entered in text fieldof dialog boxand information which have been previously supplied by the software developer to the Workbench about the standard or initial environment in which the software developer's Projects are to be run. Information such as the human language (English, German etc.), default color scheme, and time zone may be included.

[0053] If, however, the Project is to interact with one or more external objectsthe software developer would activate the "next"buttonon the Project Identity Dialog Boxand an External Object Connection Server Dialog Boxwould be displayed as seen in Fig. The External Object Connection Server Dialog Boxidentifiesthe location where each external object, or group of external objects, to be referenced is located. External Object Connection Server Dialog Boxhas a Connection Text Fieldfor entry of the location of one or more of the objects to be added to the Project. Any standard form of identification, such as a Universal Resource Identifier ("URI"), may be used. A buttonmay be used to aid navigation of the available environment. External Object Connection Server Dialog Boxalso has text fields for

the user nameand passwordif needed. Where different user names and/or passwords are necessary for entry to two or more objects at the same location, each object must be entered separately with a more exact location indicated so that the user identification and password can be separately designated.

[0054] Protocols are established methods for communicating with external objects, *i.e.*, established ways for interrogation and integration of external objects, including the Internet InterORB Protocol ("IIOP"), Lightweight Directory Access Protocol ("LDAP"), Remote Procedure Call ("RPC"), Remote Method Invocation ("RMI"), Java Database Connectivity ("JDBC")and numerous others. The protocol is normally designated for each object by the platform's Naming and Directory Service. The Workbench learns about the object through introspection or reflection. First, the Workbench determines the protocol used to communicate with the external object. The software developer can specify identification of the object's protocol in text field 321along with the location. For example, entry of iiop://localhostin text fieldwould designate all objects on the client computer (localhost) at port 900 using the IIOP protocol. The port number is only required if the protocol is not located on the computer's default port. The designation localhost can be replaced with other identification means, such as its numeric equivalent. The external object reference informationand the user name and passwordallow the Workbench to investigate the objects contained at the designated location.

[0055] Once the Workbench has determined the protocol, it queries the external object to determine all its members, i.e., its methods and fields. The software developer is then given the opportunity to select which of these fields and methods the developer wishes included in the final application. Activation of the "next"button 324 causes the Field Selection Dialog Boxto be displayed as seen in Fig.6. The Field Selection Dialog Boxallows the software developer to decide whether to include all fields or methods of the external objects just selected by means of External Object Connection Server Dialog Boxor only some of these fields. In the left areaof the Field Selection Dialog Boxis a listing of the external objects just selected and the right areadisplays all available fields and methods

associated with the external object selected in area 341. Both the objects and the fields can be marked for inclusion, or unmarked to be excluded from the Project by clicking on the external object or field.

[0056] If additional objects are to be included from other locations, other objects having different protocols are to be specified or if different user names or passwords are necessary with regard to certain of the objects, the software developer activates the "next"buttonof Field Selection Dialog Boxand a new External Object Connection Server Dialog Boxis displayed in which fields are selected by Field Selection Dialog Box 340 as above. This process is repeated until all additional objects, protocols, locations, user names or passwords have been designated, whereupon the software developer activates the "generate"buttonor 344.

[0057] When the software developer activates the "generate"buttonor 344, the Workbench generates the Project specification file or Base Projectsimilar to that generated for a Project which does not interact with an external object. For Projects interacting with external objects, there are general references to the external object previously enteredsuch as the server connection for each object, user ID, password, in the Project Specification File. For each method and field in a designated external object, the Workbench generates by reflection a list of attributesincluding its class name for inclusion in the Project specification file gathered from the external object.

[0058] Figure 7 is a flow diagram of the generation of the information about each external object for inclusion in the Default Project. The information will be sufficient for the creation of a window for each external object.

[0059] An action is a component which causes such a window to be displayed in the application. The Workbench generates the information about an appropriate action for each external object. The Workbench gives the action the same name as that which designates the external object and provides default parameters. That action may be attached to a tool bar, menu or button during the editing stage. The Workbench next generates the window's properties such as size, color, and font,

using default values. It uses as the names of the window, the name of the external object and thus, the action. The Workbench thereby associates the window with the external objectWithin each window, one or more panels is generated. The panel is a connection between the external object window and the external object to which it relates and is connected to the external object upon building of the application by the Runtime. Each panel is associated with, i.e., connected to, only one external object. Multiple panels can be placed in a window. The Workbench, by querying the external objects, provides the information for such connection within the Default Project.

[0060] The Workbench then looks at each accessor method and field in the external object and generates the corresponding description for text fields, combo boxes and radio buttons for each using default values for each such panel type The nature of the panel depends on the nature of the accessor method or field. For example, The Workbench would produce descriptions of radio buttons or combo boxes for range bounded data, check boxes for set data, and text fields for data that is neither range bound nor set in nature. Text field sizes are set according to the kind of data that is expected and may be masked to further complement the data display and input. For each accessor method that is identified on the external object, a text field is generated the size of which is set according to the type of parameter or return object that is declared.

[0061] The Workbench develops a label for each text field, combo box and radio button. The text of the label corresponds to the name of the accessing method or field.

[0062] If, there are any members (method or data) that could not be associated with text fields, radio buttons, combo boxes, the Workbench creates for these members an unreferenced member listto aid the software developer. The software developer can use information contained on the unreferenced list to further extend the behavior of the project via programming code during editing. That is, the software developer can utilize the members from the unreferenced list to add additional functionality to the Built Project.

[0063] The Workbench has combined the information about the external object gained from reviewing each component and sub-component of the external object and combined this information with certain default properties in order to develop a complete description of an external object window allowing access and use of all selected components of the external object. Once a Project Specification File is completed, the Workbench generatesa Project directoryshown schematically in Fig. Into this directory, the Workbench loads the Project Specification File 334 and three empty sub-directories; one for image files 331, one for help filesand one for programming code files If an application were to be generated using the information obtained in the Default Project, the Runtime would generate a frame or dialog for each object having a text field for each field and a text field for each accessor method. The frames would then be listed in Project Box 304.

[0064] Obviously while an application arising from such a default Project could be run, refinement of this default information gathered by the Workbench would in most cases create a far more useful application. From this point forward, the Workbench acts as editor of the information about the selected objects in order to refine the ultimate display of the interaction with the object.

[0065] Editing of the Projectbe accomplished either on a newly generated Default Project or by selecting and loading an existing Project. To select an existing Project, the software developer either from the file menuor from the folder iconon the tool baropens an Existing Project Dialog Box As seen in Fig.an Existing Project Dialog Boxcontains a directory navigatorfrom which the software developer selects a Project in the normal manner. The Workbench loads the selected Projectsupon selection of the open button This process can be repeated to load several Projects.

[0066] Once a Project is opened using a dialog boxhierarchical listis displayed in Windowof the major components of the Project as seen in Fig. These would normally include for each project, external objects, windows, components, styles, and notes which were identified in generation of the Default Project. If multiple projects have been opened, each hierarchical list will be made visible by clicking on the appropriate Tab 401.

[0067] Each element of Listmay be opened to the next level by clicking on the appropriate icon Once an element to configure is selected, its pertinent parameters are displayed in property window These initial values were set during generation of the Default Project as discussed above. A new element is added at any level by right clicking on the listing onin window to which it will be a component. As seen in Fig. 11, for example, the top level Windowshas been opened to show two sub-levels, Dialogs 407 and Frames 408. A new Frame 419 has been added and been titled through text field 410 in New Frame Dialog Box 409 generated by right clicking on Frames listing 409. As seen in Fig. 12, standard properties for the new frame are displayed in the properties window 305. List 411 itemizes various properties of the new item and 412 displays the values. Initial values are set automatically but may be modified either by drop down lists 413 or text fields 414. Lists 411 and 412 allow the new frame to be fully defined. They include such things for a frame as background image, tiling, the caption positioning, and the like. There is also a text field 415 for identifying a file of additional programming code to modify the behavior of the frame.

[0068] As seen in Fig. 13, after the general parameters of the appearance of the main screen 419 have been developed in the properties window 305, a panel is generated in a similar manner by right-clicking on the panels entry 420 in box 304, opening new panel identification box 430 with text field 431 for entry of the name of the panel. This in turn generates both the necessary component elements of the panels 431 and various sub-elements 432 - 434. Upon selection of any of these elements, an appropriate list of necessary elements having standard values is generated in the properties window 305. A help box may also appear 437. An edit window 500 is generated, which displays the panel 510 as it is original automatically generated and as it is modified during editing. The elements listed in the properties window 305 and in the edit window 500 are interactive. In other words, a change in box 305 will change the appearance of panel 510 and a change made directly on panel 510 will make a change in the values displayed in 305.

[0069] As most clearly seen in Figs. 16, 17 and 18, in a similar manner labels and text fields can be generated to fully form the desired panel. Through repeated

iterations box 510 will have all the necessary elements added to the panel as displayed in Window 500 shown in Fig. 18. If there is to be an interaction not automatically generated by the Workbench, a visual object may be defined and added to the panel 510. A new Visual Object Dialog Box 512 is displayed with a text field for naming the visual object as seen in Fig. 19. The properties of the visual object are defined by properties box 305. As seen in Fig.20, a new button 520 is displayed in panel 510. As shown in Figs. 21 & 22 this new button can be associated in a conventional manner with programming code to perform the desired function and saved at a location for later incorporation in the Built Project. Collections of programming code for use in the Built Project may be simply collected in a single directory or organized in subdirectories as desired by the software developer, for example, by subject matter.

[0070] After normal editing is completed, the project may be globalized 104. In this context globalization of the Project is the incorporation of alternate versions in different human languages into a single Project JAR. During globalization, the software developer provides one or more alternate entries grouped by language for each human language to be available in the Project. The Runtime determines the correct language, locale, and dialect from the settings on which the platform that the application runs. If a computer's default language is German and the Built Project has been globalized to include German, the application generated from the Built Project will be in German in a German format and respond to a German keyboard. Equally, the ability to substitute alternate elements and programming code within the Built Project allows the use of accessibility features for disabled users. Thus, for a blind user, the Built Project may allow ready substitution of sounds or Braille for the written word.

[0071] At this point the Project parameters primarily reside in the Project file as shown schematically in Fig. 8. As noted previously, the Project specification file 334 defines the Project through a list of properties and values in a compressed text file shown in uncompressed form in Fig. For example, the size of the font in a default label may be designated by its branch (style) and its sub-branches and properties (font, default label, size) as "style.font.DefaultLabel.size"560 with a value of

"12"561. The project can be tested 106 in this format by selecting the appropriate menu item under the project menu 380, or by selecting icon 381 on the tool bar. This generates an application based on the Project in its present non final state and allows the software developer to test its operation. Help files for the project can then be generated in a conventional manner.

[0072] The project is now ready for building 108. The software developer indicates by selection of the build option on the project menu 380 or the appropriate build icon 389. A Build Project Dialog Box 600 is displayed which in text field 601 allows entry of the version number and in text field 602, the location where the Built Project is to at least temporarily reside. After entry of this information, building of the project can be commenced by activating "build"button 603.

[0073] The Workbench then proceeds to use the Project Specification File 334, the image files previously collected in the image directory 331 and the help files collected in help directory 332 to generate a Built Project directory. The Project Specification File 334 listings are converted to a Project Specification Object 385. The Project Specification Object 385 is organized into the equivalent of a structure of directories and subdirectories based on the various branches and sub-branches and values contained in the Project Specification File 334. By saving the information in the Project Specification File 334 as a Project Specification Object 385, the need for the Runtime to convert the compressed text file of the Project Specification File 334 with the corresponding loss of time is eliminated.

[0074]

The image files that were contained in the project file directory 331 are stored in the Built Project Image Directory 381 shown schematically as Fig. 25. Similarly, the help files contained in the Project Directory 332 are stored in the Built Project Help Directory 382. The programming code files that may have been attached to one or more aspects of the project 383 may at this time also be stored in the Built Project Directory 382. The programming code files collected in the Built Projectmay be organized in the same fashion as it was organized and saved during generation. Finally, a manifest 384 is generated containing general information about the Built Project, such as identification of the software developer, version number, the

starting instruction which begins the development of an application from the Built Project, an indication of additional JARs necessary to build the application and whether the JAR is locked such that the JAR may not be altered.

[0075] The resulting Built Project can be used with the Runtime to generate the desired application. The Built Project and the Runtime in combination build the application which then runs independently of the Runtime. The Runtime may be written in Java and contained in a JAR 850 shown schematically in Fig. 26. As such, it contains a standard .obj 851, Class files 852, Reflection Engine 853, a Support Library 854, and a Manifest 855. The Reflection Engine 853 is the heart of the Runtime. It provides the logic that is necessary to interpret the Built Projects contents, reflect upon the related environment and produce the user interface, and connect the user interface to the external objects that are specified. The Standard.obj 851 contains specifications that are common to many projects in a Serialized Specification Object similar in construction to that of the Project Specification Object 385 of a Built Project 380. The Support Library 854 is a collection of executable programming code that is used to assist the software developer by providing common development elements not part of the Reflection Engine. Many elements of the Support Library are coupled to standard.obj specifications. The Class Files 852 are executable files that are not part of the Reflection Engine or Support Library that assist the execution of the Project, such as files to assist in bootstrapping the Built Project 380. The Manifest contains information about the construction of the Runtime. Together they form a program to build an independent application based on the Built Project 380 using the JRE already residing on the computer.

[0076] The Runtime Reflection Engine 853 constructs the application from the Built Project, i.e., it constructs all of the objects that compose an application that is described in the Built Project. As noted above, when the Default Project is generated a description of each field and method is added to the project specification file 334 which upon finalization of the project becomes part of the Project Specification Object. The Reflection Engine first looks 1500 at the Project Specification Object 385 to determine each component underlying class

designation. It then maps the construction features 1501 set forth in the Project Specification Object 385 for the component needed to be specified for that class to help the runtime properly reflect upon the target class. It uses information contained in the Built Project 380 including the target class, the methods that are expected, and a mapping of common method names to variants of those names. This step is necessary to resolve individual differences between the interfaces of commonly used classes. The Reflection Engine then constructs a visual object based on this information by using as a constructor the constructor specified in the Project Specification Object 385 or if none is specified, the default constructor for that particular Java class. The Reflection Engine then maps operation features 1503 for use in connection with the operators and then installs the post instantiation operators such as listeners to respond to events such as caused by clicking on a visual object.

[0077] Thus instead of the software developer having to know the specific constructor for the object to be generated, the Reflection Engine looks to the class of the object and thus the constructors available for such object and either selects one specified in the Project Specification Object, or selects the default constructor for that class. By use of the Reflection Engine, Built Projects may also easily be retargeted without the need to readdress the programming of the Built Project. This feature becomes important using the same application on devices that have widely varying capabilities. For example, a Built Project may be created for a full featured platform like a desktop but could then be deployed on a cellular telephone. The cellular telephone environment may have a significantly different set of environment classes.

[0078] In operation the Runtime first queries the user's computer as to standard settings, including language, physical location systems and the like to determine aspects of the completed application Thus, it may determine that the machine is located in Germany and uses the German language by a review of the existing computer files and accordingly would build the resulting application in German using language formatting of a keyboard appropriate to a German user. If for some reason the Runtime cannot determine the language setting of the user's computer

or the Built Project was not designed to support the computer's default language, the Runtime will use the Built Project's default language.

[0079] The Runtime takes the information in the Built Project 380 and deserializes the Project Specification Object. The Runtime then determines if there is any programming code 383 to be used in connection with the application and if so loads and invokes any open methods1007. Open methods are methods that are defined by the Built Project initiated prior to the application becoming available to the user such as preparing a log file.

[0080] The Runtime is now ready to determine if the application will have the standard look and feel of the user's computer platform, or special look and feel features in the Built Project 380. If the look and feel is not specified by the Built Project, the resulting application assumes the computer platforms look and feel. If, however, the look and feel is specified, the runtime attempts to load the look and feel into the application. If, for some reason, the look and feel that is specified is unavailable (not present or not licensed to run on the target platform) the default look and feel for the user's computer platform will be used. At this point it may also see if there are any special accessibility features and if so, initiate such features 111.

[0081] The Runtime installs the Help 1012 and creates the desired application desktop based on the parameters supplied in the Built Project JAR 380 using the information in the Project Specification Object 380. The Runtime then connects the application so built to any external objects 1014, displays the specified splash screens 1015 and 1016 and displays any start window 1017 and 1018.

[0082] The result is an independent application which stands free of the Runtime. Thus, both the Default Project and the Built Project are in actuality abstractions of the application. That is they contain the abstract concepts delineating the desired application stored in the form of object that are related to one another. These objects and relationships are maintained in the Project Specification File and consequently the Project Specification Object 385 for Built Projects. The Runtime takes this abstraction and creates the application. The user in using this

application does not in any way reference the Runtime or in fact interact with the Runtime after the application is built. Thus, for any given use of the application, communication with the Runtime has essentially ended after the application has been constructed.

[0083] Given the nature of intranets and the Internet, the Built Project Runtime and external objects may reside on the same or different computers in such networks. For example, as shown schematically in Fig. 29 the Built Project 380 and the Runtime 850 may reside on the user'ss client computer 1200 while the external objects 1202 to which it interconnects may reside on the server 1205. Equally, the Runtime 850 may reside on the server 1205 with the external object, leaving only the Built Project 350 on the client 1200. Similarly, as shown in Fig. 31 the Built Project 380 and the external objects 1202 may reside on the server 1205 with only the Runtime 850 residing on the us's computer 1200. The simplest alternative is for all three, the Built Project 380, the Runtime 850 and the external object 1202 to reside on a single computer 1206. These alternative constructions are possible since both the JAR containing the Runtime 850 and the JAR containing the Built Project 380 are of relatively small size with the JRE already residing on the user's computers. Other combinations using middleware and relating to multiple external objects may lead to alternative configurations.

[0084] It is understood that the present embodiment described above is to be considered as illustrative and not restrictive. It will be obvious to those skilled in the art to make various changes, alterations and modifications to the invention described herein. To the extent that these variations, modifications and alterations depart from the scope and spirit of the appended claims, they are intended to be encompassed therein.